16-BIT OPERATING SYSTEMS, A WHOLE NEW BALL GAME

A critical component in the success or failure of any 16-bit computer is the operating system

by Andrea Lewis

It is obvious from the increasing amount of material written on operating systems in recent months that choosing an operating system has been recognized as central to what the system, as a product, will ultimately become. While 16-bit microprocessors have been available since Intel first introduced the 8086 over five years ago, only now are they finding their way into the general systems marketplace. This is because the standard operating environment enjoyed by 8-bit systems has been lacking. With no standard environment, cost-effective applications could not develop.

Two years ago it became apparent that there were no suitable candidates for the operating system portion of the 16-bit standard operating environment. Digital Research was translating CP/M-80 and MP/M-80 to run on the 8086. However, facilities that the emerging 16-bit technology would require were not provided, and the problem remained unsolved. Also, standards on 8-bit machines did not automatically become standards in the 16-bit market.

Part of the confusion in much of what has been written indicates that there is a lack of understanding regarding the nature of operating systems and the differences between 8- and 16-bit systems. While these differences are obvious to a technically versed reader, engineers who select a 16-bit operating system because it looks like the 8-bit standard operating system evidently do not consider the ramifications.

Andrea Lewis is manager of technical publications at Microsoft, 10700 Northup Way, Bellevue, WA 98004.

Unique 16-bit market

The 16-bit market is a whole new ball game—16 bit microprocessors are totally incompatible with their 8-bit predecessors. Not even compatible operating systems will allow software written for an 8-bit microprocessor to automatically run on a 16-bit machine. As languages such as assembly are specific to the instruction set of the chip they run on, application programs written in assembly language have to be rewritten. Mechanical translation is a good first step in software migration, but for complex programs the resultant software must still be substantially re-engineered to take full advantage of the 16-bit processor's added capabilities.

Application programs, written in a high level language, port easily once the language has been rehosted on the new machine. The language itself minimizes operating system dependencies in applications. An 8-bit operating system's part in forming the standard operating environment on smaller machines was to support the languages. It actually made little direct contribution to the substance of the application.

Advanced human engineering is one of the 16-bit system's promises. In the areas of color graphics, pointing devices, and other forms of nonkeyboard input, expectation for advancement runs high. The only way portable applications can be written to take advantage of these hardware capabilities is if the software that underlies the applications—languages and operating systems—offer the facilities needed.

Two distinct but merging markets

The 16-bit microprocessor's increased power and flexibility allow it to address a broader range of applications than 8-bit chips. Presently, the two types of 16-bit systems fall at opposite ends of a spectrum. High end

TABLE 1

Operating System Contenders

Multi-User Micro Mainframe	Low End Personal Computers
UNIX-derived operating systems (eg, XENIX)	MS-DOS
UNIX-like operating systems (eg, Chromix)	CP/M-86
MP/M-86	
Oasis-86	
RMX-86	

microcomputers are designed to meet the small to intermediate minicomputer offerings head-on. The term "micro mainframe" is often used generically to describe this class of machines. This reference should not be confused with proprietary products similarly described.

8-bit microcomputers made several feeble attempts to challenge this market with 8-bit versions of MP/M and Oasis. Introduction of the 16-bit microprocessor and more sophisticated systems software, however, will eventually lead to a serious clash between mini- and microcomputers. At the low end of the broadening 16-bit spectrum are personal computers, built as supercharged entries into the market created by Apple II and TRS-80. As seen in Table 1, the high end of the range has already been heavily populated with contenders for the standard operating system. There are really only two offerings at the low end—MS-DOS and CP/M-86.

Micro mainframe characteristics and applications

Ideal machines for the high end of the 16-bit market have capabilities previously found only on minis from companies such as Digital Equipment Corp and Data General. Supporting a respectable number of users in a true timesharing environment, these systems have large amounts of main memory and hardware facilities for memory management and protection, and require hard disks. Generally, they have a tape drive available for backing up large amounts of data stored on disk. Some of the less expensive systems use floppy disks for backup. Starting in a price range of $12,000 to $15,000, they offer performance comparable to that of minicomputers priced at several times that range.

The more sophisticated end of the 16-bit marketplace addresses many of the same applications formerly handled by minicomputers. Expanded memory and processing power of 16-bit systems allow them to be used in many highly demanding applications environments. Until now, the minicomputer has owned the computer aided design/computer aided manufacturing industrial control, and network controller markets. Now, however, many companies are effectively competing with the minicomputers with 16-bit microcomputer systems that utilize XENIX or some other UNIX variant.

To challenge the minicomputer, the micro mainframe must offer the mini facilities that have evolved over the last 10 years. Putting a minicomputer operating system on a toy computer will not result in a cheap minicomputer. To support multiple users in a true timesharing environment, reliable memory protection must be provided to keep one errant user from disrupting the other users' work on the system. Dynamic memory allocation gives each user access to the machine's full memory capacity, rather than segmenting each into an isolated partition.

I/O subsystem

It is often assumed that a 16-bit microcomputer's performance depends primarily on the choice of microprocessor and the efficiency of the operating system. However, memory management and the input/output (I/O) subsystem are areas where system architecture seriously impacts system performance.

I/O architecture is perhaps the most important and most often overlooked part of system design. All I/O must be interrupt driven. To suspend the first task and run the second, while the first waits for I/O to complete is an unacceptable burden for the system, if the system must loop waiting for the I/O.

A system that supports 10 terminals at 9600 baud would be allowed around 100 μs/interrupt. A processor capable of one instruction in 2 μs would then be allowed only 50 instructions to process each interrupt. Saving context, processing character, and restoring context with just 50 instructions is infeasible. Direct memory access (DMA) can be used to reduce the interrupt burden, as can a dedicated I/O processor. However, an added processor only reduces the burden—it does not eliminate it. Even if an I/O processor handled all the interrupts in the above example, only 50 instructions would be available to process each character.

Many larger computers suffer from inadequate I/O bandwidth. While there are no easy rules to guide this design, the following calculation can put bounds on a design. If a system requires 1000 instructions to process an interrupt and the target is to keep the burden of handling these interrupts at a 50% overhead, the 500,000 instructions/processor can handle at most 250 interrupts/s (50% of 500,000 = 250,000; 250,000 instructions/s, and 1000 instructions/interrupt).

Even when DMA devices are provided, a multiprogramming system's overall performance is typically bound by the performance of the mass storage devices on the system. This is especially true when storage is used for swapping or paging. Such a system can be expected to demand transfer rates on the order of thousands of blocks/s, just as the performance of the character devices is improved by providing multisector transfer capability.

Common operating system requirements

Common requirements exist for both types of 16-bit systems. Both require more capable operating systems than exist for 8-bit machines. These operating systems are needed as a foundation for advanced languages and applications. Both require standardization of operating systems so software that needs to interact heavily with the operating system will have a broad base of machines to move into. Both systems also require standard languages and utilities to provide a large contiguous market for applications software.

As much as these markets need software sophistication, it is clear from the comparison of system requirements in Table 2 that one operating system cannot support the full range of computers. Diversity in hardware capabilities present at each end of the spectrum and the degree of sophistication needed make it

TABLE 2

Comparative System Requirements

Operating System Characteristic	Low End Personal System	High End Micro Mainframe
Size of OS	Under 15k	Up to 100k allowable
Number of users	Single user	Up to 16 (or more)
Main memory	As little as 48k	To 16M byte
Mass storage	As little as 160k	5M byte minimum
Utility support	Minimal	Extensive
Runtime support	Simple file server	Demanding requirements

impossible. That is why the market is served by systems as diverse in capabilities as MS-DOS and XENIX.

XENIX addresses high end market needs

XENIX offers all the capabilities of any leading minicomputer operating system, plus software development tools and functionality not provided on non-UNIX systems. Clearly, XENIX could not provide this utility in the limited hardware environment of the low end personal computer.

UNIX was the logical choice as an operating system base for Microsoft's high end operating system. UNIX is portable, written 99% in the C programming language. Its simple design allows great flexibility. Well-known for its powerful software development environment, UNIX also has a user interface that can easily be tailored. This interface allows persons building turnkey applications to provide a friendly profile without having to build the interface into each program. UNIX's flexibility becomes apparent to many engineers using UNIX for the first time when they find they do not need to modify the operating system for this tailoring.

Tailoring the user interface

The shell is UNIX's command line processor—a powerful, high level programming language in its own right. It allows for control statements, parameter passing, variables, and string substitution. It also has such programming constructs as while, if then else, case, and for. With all UNIX programs, a shell can accept input from any specified source, and provide output to any destination.

Because the shell is often referred to as *the* shell (implying that there is only one) persons unfamiliar with UNIX sometimes miss the fact that the shell is just another program. Thus, there can be multiple shells on the system. Because UNIX is fully multitasking and recursive, each user can run under a unique shell to tailor the environment each individual on the system sees. It is not unusual for users running on a well designed applications system to move from one shell environment to another as tasks change. This eliminates, or simplifies, the need for applications programmers to write lengthy programs to control user interface. Moreover, it encourages the development of applications systems consisting of small, efficient, and easily maintained routines rather than monstrous applications programs.

The shell allows custom command implementation on the system. When the standard UNIX shell receives a command, it normally searches the user's current directory for an executable file with the name of the command issued. If one is not found, it then checks for a similar file in the user's special command directory. If a match is still not found, the system's command directory is searched. Characteristic of UNIX, this search path is also readily modified to allow multiple command directories, or to optimize the search to increase the probability of a correct match on the first search.

For example, UNIX provides an electronic mail facility, invoked by typing "mail." It is not unusual for a user to want to save mail, or to keep an ongoing record. Sometimes it is convenient to file these messages in a separate directory for later action rather than save them in whatever directory the user has active when mail is invoked. Assuming a directory (mail.dir) has been established to hold this mail for future action, a shell script in the file named "readmail" in the user's command file would take the user from whatever directory was currently defined as the working directory to the mail directory (mail.dir in this example), execute the mail program (an example of UNIX's multitasking), and in effect, return the user to the working directory when the readmail command was invoked.

Device independence and recursiveness

The UNIX system's generalized, modular design is consistent throughout, giving it flexibility. During its evolution, each task that developers envisioned the operating system would perform was broken down to its simplest form and implemented as a separate capability. Since I/O is totally device independent in UNIX (ie, a program's source of input and destination of output can be dynamically changed without change to the program itself), these individual tasks can be linked together as needed, even recursively. This allows a programmer to apply them freely to the problem at hand. Therefore, operations can be performed that may have never been envisioned by the system's authors.

I/O in the UNIX system is considered as a stream of bytes, making a simple and clean interface between UNIX and system devices. It also makes it possible for the complete freedom of I/O redirection, fundamental to the UNIX environment. This byte-stream orientation is often misunderstood by traditional applications programmers to mean that there can be no concept of record oriented I/O on a UNIX system. Actually, nothing could be further from the truth—it allows a programmer to define whatever type of file structure is required. For applications programmers who have no desire to write code to implement such structures, high level languages being adapted to run under UNIX (BASIC, COBOL, etc) implement their own traditional sequential, random, and indexed file structures.

Another component of device independent I/O implementation on a recursive system is the standard input and output concept. (See Fig 1.) At any given time, each

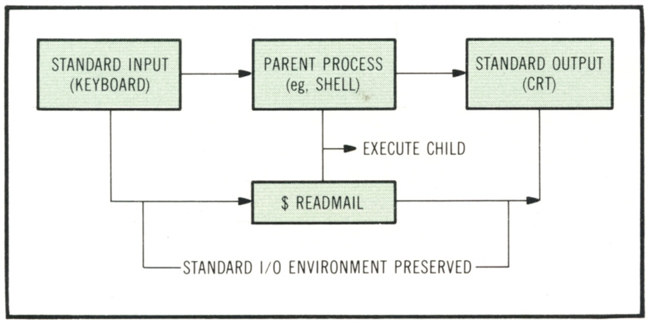

STANDARD INPUT (KEYBOARD)
PARENT PROCESS (eg, SHELL)
STANDARD OUTPUT (CRT)
EXECUTE CHILD
$ READMAIL
STANDARD I/O ENVIRONMENT PRESERVED

Fig 1 Simple child execution exemplifies how standard I/O provides device independence.

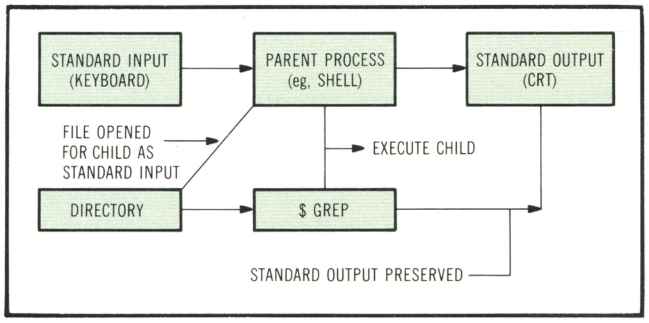

STANDARD INPUT (KEYBOARD)
PARENT PROCESS (eg, SHELL)
STANDARD OUTPUT (CRT)
FILE OPENED FOR CHILD AS STANDARD INPUT
EXECUTE CHILD
DIRECTORY
$ GREP
STANDARD OUTPUT PRESERVED

Fig 2 Grep feature can be redirected to allow for new input standards by use of symbol.

task running on the system will have a source of input defined as its standard input and a destination defined as its standard output. For example, the shell typically has the keyboard defined as its standard input; the cathode ray tube (CRT) as its standard output (although either or both can be redefined). If a child process is spawned (eg, the invocation of the mail program in the above example), it automatically inherits the standard I/O definitions of its parent process. Therefore, unless I/O is further redirected (which can be done freely), if a process (readmail) using the terminal for I/O spawns a child (mail), that child will also use the terminal for I/O unless otherwise specified.

UNIX provides superior utilities

UNIX flexibility, and the authors' philosophy of adding capabilities in a modular form, has stimulated the development of unusual yet useful utilities not found elsewhere, such as grep. Grep scans a source of input for any string fed to it on invocation and returns records containing matching strings, or if desired, a count of how many records contain that pattern. Grep is obviously useful for programmers editing and modifying sources, but it is useful for others, too. One business oriented use of grep is to compile marketing tallies from information request coupons. Another frequent nontechnical use of grep is for a person to build a file of names, addresses, and phone numbers of frequent contacts. Grep is then used to retrieve appropriate information as required. Normally, grep would inherit standard input from its parent process—in this case the shell—which would be the keyboard. However, the character "<" directs the parent to set up a new standard input for the child—in this case, a file named "directory." (See Fig 2.)

To make access to the phone directory convenient, users take advantage of the ability to create a shell script that accepts an argument at execution and passes that argument to grep, which searches the directory file and returns the record or records containing that argument (see Fig 3). To retrieve a phone number, the user types the file name phone and a name, company name, or something to identify the person to be retrieved at that particular time (see Fig 4).

As a final example, the UNIX system can automatically feed one program's output or utility into the input of another utility or program. This is called a pipe. Through the use of pipes, various utilities provided with the system can be joined together to suit a particular purpose. Any number of utilities and/or user programs can be joined with pipes—the same program can even appear more than once in the pipeline (recursively). Fig 5 shows the output of grep being piped to a program called tr, which converts colons separating portions of the records into carriage returns and linefeeds for formatting purposes. Then data are passed to a program called pr, which prints the data (in this case to a file called phone.out) in a page format with date/time headers. (See Fig 6.)

```
One Line Shell Script in file named "phone":

                grep $*<directory
                :    :  :
                :    :  :
                :    :  :..>Name of File to Search for Pattern
                :    :....>Shell Substitution Variable
                :..>XENIX Utility to Search a File for a Pattern

Sample Execution Using the "phone" shell script to find a given
person:

$ phone Wood
:  :     :
:  :     :
:  :     :..>Text String to Search for in the File Named "directory"
:  :....>Name of File Containing the Shell Script
:..>Shell-Provided Command Line Prompt (Equivalent to: "A>" in CP/M)

String Returned:

Steve Wood: Microsoft: 10700 Northup Way: Bellevue, WA 98004:
(206)828-8080

Another Example to Find all Entries in Bellevue:

$ phone Bellevue

Mark Deutsch: Microsoft: 10700 Northup Way: Bellevue, WA 98004:
(206)828-8080
Steve Wood: Microsoft: 10700 Northup Way: Bellevue, WA 98004:
(206)828-8080
```

Fig 3 Users can create shell script to aid in data searches. Here, phone directory file is searched.

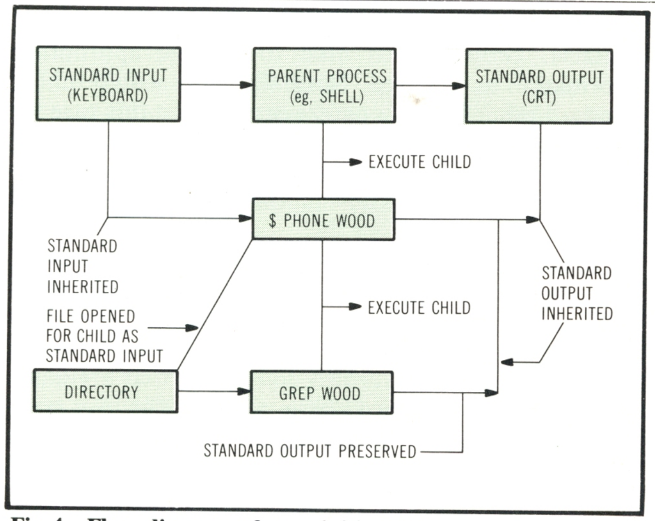

STANDARD INPUT (KEYBOARD)
PARENT PROCESS (eg, SHELL)
STANDARD OUTPUT (CRT)
EXECUTE CHILD
$ PHONE WOOD
STANDARD INPUT INHERITED
FILE OPENED FOR CHILD AS STANDARD INPUT
EXECUTE CHILD
STANDARD OUTPUT INHERITED
DIRECTORY
GREP WOOD
STANDARD OUTPUT PRESERVED

Fig 4 Flow diagram of grep initiated data search

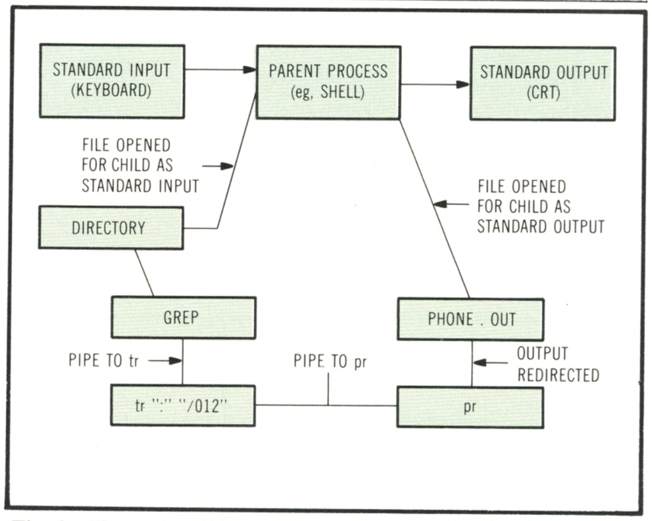

STANDARD INPUT (KEYBOARD)
PARENT PROCESS (eg, SHELL)
STANDARD OUTPUT (CRT)
FILE OPENED FOR CHILD AS STANDARD INPUT
FILE OPENED FOR CHILD AS STANDARD OUTPUT
DIRECTORY
GREP
PHONE . OUT
PIPE TO tr
PIPE TO pr
OUTPUT REDIRECTED
tr ":" "/012"
pr

Fig 6 Flow diagram of piping procedure

Weaknesses of UNIX

Standard UNIX, as it comes from AT&T, does have its failings when examined in its role as a commercial microcomputer operating system. Supplied by AT&T, UNIX is configured for use on the DEC-11 and VAX lines of computers, and is aimed at the typical data processing environment. Therefore, the system was implemented with the assumption that a systems programmer would be available to act as administrator to boot the system up, take it down, etc.

Developing UNIX in a research environment also means that it was considered less important to worry about things like imperfect disk media. File system reliability was not a concern, as a systems programmer was always available to make things right. Documentation was written with the assumption that users had access to source code and knew what to do with it.

From the standpoint of micro implementations of UNIX, an additional problem is that UNIX is tuned to the PDP-11 world where disks are fast. Just the opposite is true on a micro—Winchester disks are fast compared to floppies, but crawl in comparison with PDP-11 drives. Consequently, UNIX's scheduling and swapping mechanisms are rather single-minded because they rely on high performance disk drives.

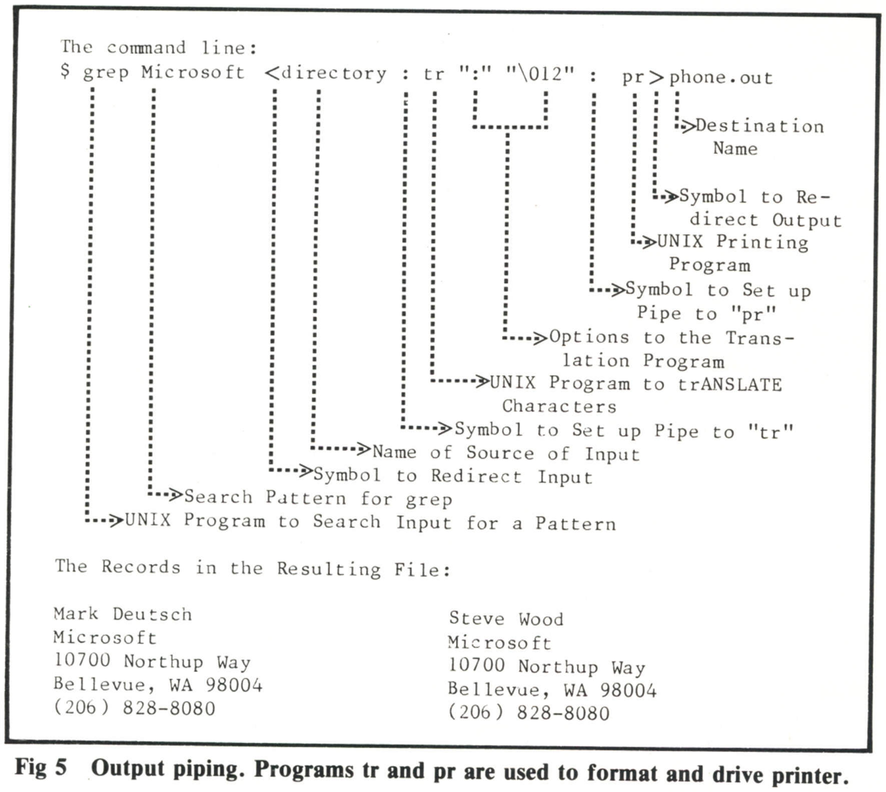
A classical timesharing system, the UNIX system is inappropriate for highly demanding realtime systems requiring guaranteed immediate response to a given event. Realtime limitations are inconsequential in general data processing, but with the increasing importance of communications, especially local networks, these deficiencies become serious. This is one of the key areas where the company is focusing XENIX development.

XENIX enhancements

A popular misconception must be clarified, however, before discussing some differences between XENIX and UNIX—XENIX is not a UNIX lookalike. XENIX was developed from the UNIX system and is sold under license from AT&T.

It takes a significant effort to turn a research oriented minicomputer operating system into a general purpose, commercially viable product, even if it is portable. Table 3 summarizes improvements and enhancements made to UNIX by XENIX. The most obvious advantage to working with XENIX is that it has been ported. When implementing XENIX on a Z8000, 8086, or 68000, it is begun with XENIX for the Z8000, 8086, or 68000.

To improve the UNIX file system, several features and utilities were added, and general design modifications were made. These allow the system to automatically recover the file system from a crash, performing functions that designers assumed would be handled by a systems programmer. In addition, XENIX was made more intelligent in its use of



```
The command line:
$ grep Microsoft  <directory : tr ":" "\012" :  pr>phone.out

                                                   >Destination
                                                     Name
                                                 >Symbol to Re-
                                                   direct Output
                                               >UNIX Printing
                                                 Program
                                             >Symbol to Set up
                                               Pipe to "pr"
                                    >Options to the Trans-
                                      lation Program
                          >UNIX Program to trANSLATE
                            Characters
                      >Symbol to Set up Pipe to "tr"
                >Name of Source of Input
            >Symbol to Redirect Input
    >Search Pattern for grep
 >UNIX Program to Search Input for a Pattern

The Records in the Resulting File:

Mark Deutsch                           Steve Wood
Microsoft                              Microsoft
10700 Northup Way                      10700 Northup Way
Bellevue, WA 98004                     Bellevue, WA 98004
(206) 828-8080                         (206) 828-8080
```

Fig 5 Output piping. Programs tr and pr are used to format and drive printer.

TABLE 3

XENIXization of UNIX

XENIX Enhancements	XENIX Improvements
Record and file locks	XENIX has been ported to Z8000, 8086, 68000
Semaphores	Automatic file system recovery
Absolute priority assignment	Performance tuned to micros
Scatter loaded kernel	Device error logging
Nonblocking reads	Interactive system reconfiguration
Synchronous (blocking) writes	Miscellaneous bug fixes
Enchanced interprocess communications	

swapping, recognizing the flipflop of performance concerns between the PDP-11 and the typical microcomputer environment. Advanced memory management techniques were also employed on systems with page-mapped memory management units to minimize the need for swapping. XENIX has the ability to log hardware errors so that original equipment manufacturers can write diagnostic software to warn a user when hardware reliability is in question. Again, recognizing that XENIX is going to be used in an environment devoid of systems programmers, it was made easier to configure new versions of XENIX—necessary when adding new devices or changing certain system parameters.

As well as improvements in the existing system, enhancements provide additional functionality that applications software can take advantage of. Initial XENIX enhancements were designed to improve XENIX's usability in database and distributed computing environments.

Conclusion

Introducing a product to address the high end of the 16-bit market (micro mainframes) is as complex from the standpoint of hardware development as it is from software. Hardware for these new systems is substantially more complex than for 8-bit systems. Manufacturers can no longer put a chip on a board with a bit of memory and call it a computer. Understanding memory management issues and multi-user systems design is the primary difficulty 8-bit manufacturers have in the 16-bit market.

Serving the high end of the 16-bit microcomputer spectrum, XENIX underlies a standard operating environment uniquely suited to the many demands made by diverse systems and applications that address that portion of the market's needs.

Reprinted with permission
from Computer Design—June, 1982 issue
Copyright 1982 by Computer Design Publishing Co.